Laborator si seminar

Programare in Java si software matematic
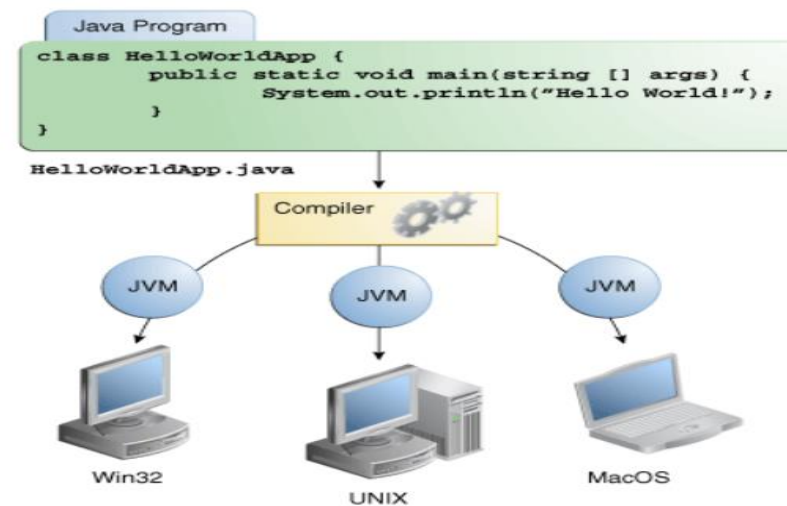
Adrian Ichimescu

Adrianichimescu@gmail.com

Java Technology

Java is a <u>high level programing language</u> and a <u>platform</u>

High level programming language means higher level of abstraction from machine language

Java

- source code is written in plain text files
- HW and OS agnostic
- It's fast (not as fast as C/C++)
- Machine code instruction can be directly executed by the CPU

Machine language

- it's 'unreadable'
- Depends on the HW
- It's fast
- Machine code instruction can be directly executed by the CPU
- load, a store, a jump, or an ALU operation on one or more units of data in CPU registers or memory

Java Platform

A *platform* is the hardware or software environment in which a program runs (ex. Windows, Linux, Solaris, Android).

Java platform is a software only platform that runs on hardware based platform.
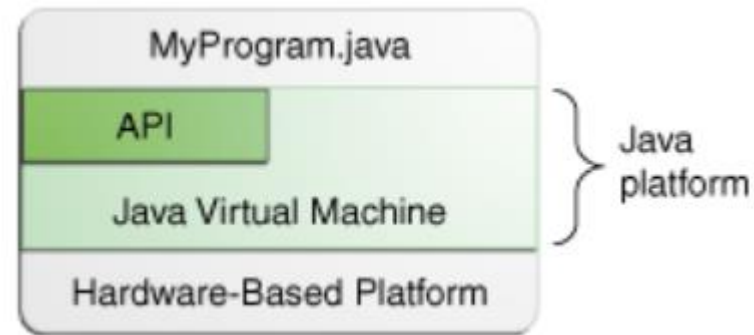
Java API is a large collection of ready-made software components that provide many useful capabilities. It is grouped into libraries of related classes and interfaces; these libraries are known as *packages*
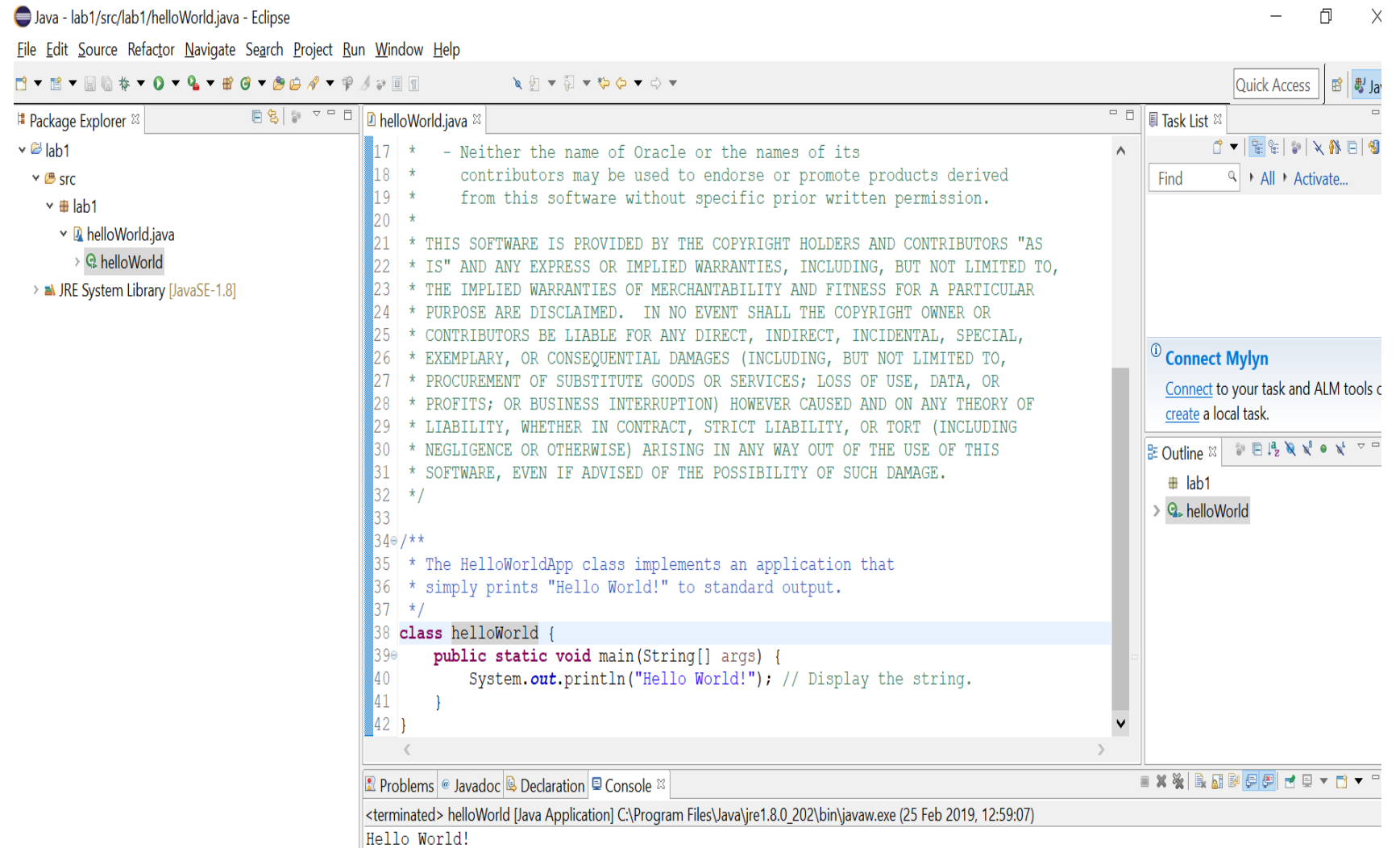
What Java can do?

- **Development Tools**: compiling, running, monitoring, debugging, and documenting your applications (ex. javac compiler, the java launcher, and the javadoc documentation tool)
- **Application Programming Interface (API)**: offers a wide array of useful classes ready for use in your own applications – details please consult the [Java Platform Standard Edition 8 Documentation](#).
- **Deployment Technologies**: The JDK software provides standard mechanisms such as the Java Web Start software and Java Plug-In software for deploying your applications to end users.
- **User Interface Toolkits**: The JavaFX, Swing, and Java 2D toolkits make it possible to create sophisticated Graphical User Interfaces (GUIs).
- **Integration Libraries**: Integration libraries such as the Java IDL API, JDBC API, Java Naming and Directory Interface (JNDI) API, Java RMI, and Java Remote Method Invocation over Internet Inter-ORB Protocol Technology (Java RMI-IIOP Technology) enable database access and manipulation of remote objects.

Java advantages

- **Write less code –** 4 times smaller source code than C++

- **Develop programs more quickly –** simpler than C++, twice faster to write a code in Java than C++

- **Write better code**: easy to extend APIs

- **Avoid platform dependencies**: You can keep your program portable by avoiding the use of libraries written in other languages

- **Write once, run anywhere**: class ->javac->machine independent bytecodes .class file->JVM

- **Distribute software more easily:** Java Web Start software

helloWorld

Type all code, commands, and file names exactly as shown. Both the *compiler (javac) and launcher (java) are case-sensitive*, so you must capitalize consistently.

helloWorld Analysis

- Comments are ignored by the compiler but are useful to other programmers. The Java programming language supports three kinds of comments:
  - /* text */
  - /** documentation */
  - // text

- basic form of a class definition is:
  class name {

  . . .

  }

- In the Java programming language, every application must contain a main method whose signature is:
  public static void main(String[] args)

  Main method it's the entry point for your application and will subsequently invoke all the other methods required by your program.

  The main method accepts a single argument: an array of elements of type String.

  This array is the mechanism through which the runtime system passes information to your application. For example:

  java MyApp arg1 arg2

- System.out.println("Hello World!");
  uses the System class from the core library to print the "Hello World!" message to standard output.

Exercise

1)

```java
/**
 * The HelloWorld class implements an application that
 * simply prints "Hello World!" to standard output.
 */
class HelloWorld2 {
    public static void main(String[] args) {
        System.out.println("Hello World!); // Display the string.
    }
}
```

Please resolve the errors!

2) Change the HelloWorld2.java program so that it displays Buna seara! instead of Hello World!.

Object-Oriented Programming Concepts

- Object
  - An object is a software bundle of related state and behavior.

- Class
  - A class is a blueprint or prototype from which objects are created.

- Inheritance
  - classes inherit state and behavior from their superclasses

- Interface
  - interface is a contract between a class and the outside world

- Package
  - is a namespace for organizing classes and interfaces in a logical manner.

Object-Oriented Programming Concepts

- Object

  An object is a software bundle of related state and behavior.

  State = fields

  Behaviour = Methods

OOP benefits:

Modularity: The source code for an object can be written and maintained independently of the source code for other objects. Once created, an object can be easily passed around inside the system.

Information-hiding: By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.

Code re-use: If an object already exists (perhaps written by another software developer), you can use that object in your program. This allows specialists to implement/test/debug complex, task-specific objects, which you can then trust to run in your own code.

Pluggability and debugging ease: If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace it, not the entire machine.

Exercise

Bicycle is an *instance* of the *class of objects* known as bicycles

```java
class Bicycle {

  int cadence = 0;

   int speed = 0;

   int gear = 1;

void changeCadence(int newValue) {

     cadence = newValue;   }

void changeGear(int newValue) {

     gear = newValue;   }

void speedUp(int increment) {

     speed = speed + increment;    }

   void applyBrakes(int decrement) {

     speed = speed - decrement;   }

   void printStates() {

     System.out.println("cadence:" + cadence + " speed:" + speed + " gear:" + gear);   }

}
```

The fields cadence, speed, and gear represent the object's state, and the methods (changeCadence, changeGear, speedUp etc.) define its interaction with the outside world

Exercise

```
class BicycleDemo {

  public static void main(String[] args) {

     // Create two different  Bicycle objects

     Bicycle bike1 = new Bicycle();

     Bicycle bike2 = new Bicycle();

      // Invoke methods on

     // those objects

     bike1.changeCadence(50);

     bike1.speedUp(10);

     bike1.changeGear(2);

     bike1.printStates();

      bike2.changeCadence(50);

     bike2.speedUp(10);

     bike2.changeGear(2);

     bike2.changeCadence(40);

     bike2.speedUp(10);

     bike2.changeGear(3);

     bike2.printStates();

  }

}
```

the **Bicycle** class does not contain a **main** method. That's because it's not a complete application; it's just the prototype for bicycles that might be used in an application. The responsibility of creating and using new Bicycle objects belongs to some other class in our application called **BucycleDemo**.

Exercise

1) What is the output running BicycleDemo ?

2) Create new classes for a real-world automobile object. Refer to the Bicycle class if you forget the required syntax.

Variables

- **Instance Variables** (Non-Static Fields) Technically speaking, objects store their individual states in "non-static fields", that is, fields declared without the static keyword
- **Class Variables (Static Fields)** A class variable is any field declared with the static modifier; this tells the compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated
- **Local Variables** Similar to how an object stores its state in fields, a method will often store its temporary state in local variables
- **Parameters** are variable of the methods and constructs

Variables

Naming conventions:
- Variable names are case-sensitive - variable's name can be any legal identifier
- Subsequent characters may be letters, digits, dollar signs, or underscore characters - When choosing a name for your variables, use full words instead of cryptic abbreviations.
- 3) If the name you choose consists of only one word, spell that word in all lowercase letters. If it consists of more than one word, capitalize the first letter of each subsequent word.
4) If your variable stores a constant value, such as static final int NUM_GEARS = 6, the convention changes slightly, capitalizing every letter and separating subsequent words with the underscore character. By convention, the underscore character is never used elsewhere.

Primitive Data Types

A variable's data type determines the values it may contain, plus the operations that may be performed on it.

8 primitive data types
- **byte**: The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive)
- **short**: The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive).
- **int**: By default, the int data type is a 32-bit signed two's complement integer, which has a minimum value of -231 and a maximum value of 231-1.
- **long**: The long data type is a 64-bit two's complement integer. The signed long has a minimum value of -263 and a maximum value of 263-1.
- **float**: The float data type is a single-precision 32-bit IEEE 754 floating point.
- **double**: The double data type is a double-precision 64-bit IEEE 754 floating point
- **boolean**: The boolean data type has only two possible values: true and false.
- **char**: The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

Array

```
class ArrayDemo {
    public static void main(String[] args) {
        // declares an array of integers
        int[] anArray;
        // allocates memory for 10 integers
        anArray = new int[10];
        // initialize first element
        anArray[0] = 100;
        // initialize second element
        anArray[1] = 200;
        // and so forth
        anArray[2] = 300;
        anArray[3] = 400;
        anArray[4] = 500;
        anArray[5] = 600;
        anArray[6] = 700;
        anArray[7] = 800;
        anArray[8] = 900;
        anArray[9] = 1000;
        System.out.println("Element at index 0: "
                + anArray[0]);
        System.out.println("Element at index 1: "
                + anArray[1]);
        System.out.println("Element at index 2: "
                + anArray[2]);
        System.out.println("Element at index 3: "
                + anArray[3]);
        System.out.println("Element at index 4: "
                + anArray[4]);
        System.out.println("Element at index 5: "
                + anArray[5]);
        System.out.println("Element at index 6: "
                + anArray[6]);
        System.out.println("Element at index 7: "
                + anArray[7]);
        System.out.println("Element at index 8: "
                + anArray[8]);
        System.out.println("Element at index 9: "
                + anArray[9]);
    }
}
```

An *array* is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.
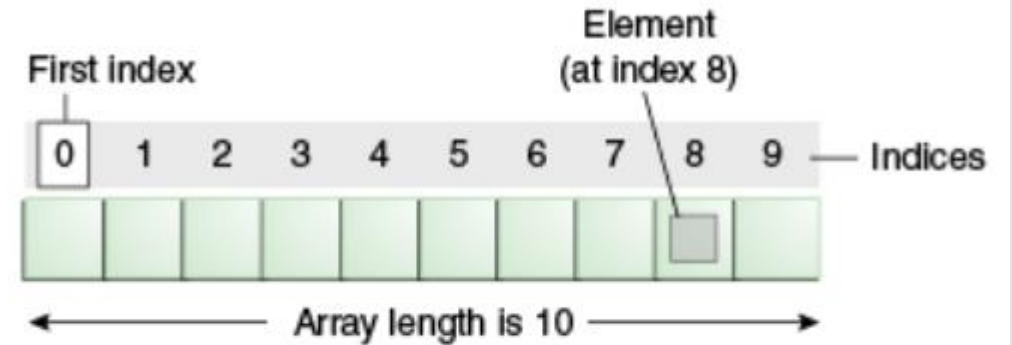
An array of 10 elements.

Exercise

1) Create an application printing the default values of a variable of any type uninitialized.